Tensorflow

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Nural Network Python Code

```
from numpy import exp, array, random, dot
```

```
training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])

training_set_outputs = array([[0, 1, 1, 0]]).T

random.seed(1)

synaptic_weights = 2 * random.random((3, 1)) - 1

for iteration in xrange(10000):

  output = 1 / (1 + exp(-(dot(training_set_inputs, synaptic_weights))))

  synaptic_weights += dot(training_set_inputs.T, (training_set_outputs - output) * output *
(1 - output))

print 1 / (1 + exp(-(dot(array([1, 0, 0]), synaptic_weights))))



from numpy import exp, array, random, dot



class NeuralNetwork():

  def __init__(self):

    # Seed the random number generator, so it generates the same numbers

    # every time the program runs.

    random.seed(1)


    # We model a single neuron, with 3 input connections and 1 output connection.

    # We assign random weights to a 3 x 1 matrix, with values in the range -1 to 1

    # and mean 0.

    self.synaptic_weights = 2 * random.random((3, 1)) - 1


  # The Sigmoid function, which describes an S shaped curve.
```

```
# We pass the weighted sum of the inputs through this function to
# normalise them between 0 and 1.
def __sigmoid(self, x):
  return 1 / (1 + exp(-x))

# The derivative of the Sigmoid function.
# This is the gradient of the Sigmoid curve.
# It indicates how confident we are about the existing weight.
def __sigmoid_derivative(self, x):
  return x * (1 - x)

# We train the neural network through a process of trial and error.
# Adjusting the synaptic weights each time.
def train(self, training_set_inputs, training_set_outputs, number_of_training_iterations):
  for iteration in xrange(number_of_training_iterations):
    # Pass the training set through our neural network (a single neuron).
    output = self.think(training_set_inputs)

    # Calculate the error (The difference between the desired output
    # and the predicted output).
    error = training_set_outputs - output

    # Multiply the error by the input and again by the gradient of the Sigmoid curve.
    # This means less confident weights are adjusted more.
    # This means inputs, which are zero, do not cause changes to the weights.
    adjustment = dot(training_set_inputs.T, error * self.__sigmoid_derivative(output))
```

```
            # Adjust the weights.
            self.synaptic_weights += adjustment

    # The neural network thinks.
    def think(self, inputs):
        # Pass inputs through our neural network (our single neuron).
        return self.__sigmoid(dot(inputs, self.synaptic_weights))


if __name__ == "__main__":

    #Intialise a single neuron neural network.
    neural_network = NeuralNetwork()

    print "Random starting synaptic weights: "
    print neural_network.synaptic_weights

    # The training set. We have 4 examples, each consisting of 3 input values
    # and 1 output value.
    training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    training_set_outputs = array([[0, 1, 1, 0]]).T

    # Train the neural network using a training set.
    # Do it 10,000 times and make small adjustments each time.
    neural_network.train(training_set_inputs, training_set_outputs, 10000)
```

```
    print "New synaptic weights after training: "
    print neural_network.synaptic_weights

    # Test the neural network with a new situation.
    print "Considering new situation [1, 0, 0] -> ?: "
    print neural_network.think(array([1, 0, 0]))
```

```
# Artificial Neural Network

# Installing Keras
# Enter the following command in a terminal (or anaconda prompt for Windows users):
conda install -c conda-forge keras

# Part 1 - Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
```

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
```

```
from keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```